Assignment 2: MLPs and CNNs in Keras/TensorFlow

Wojtek Kowalczyk
Andrius Bernatavicius

17.03.2020

Introduction

The key objectives of this assignment are:

- Learn basics of Keras and TensorFlow
- Gain practical experience with developing simple MLPs and CNNs
- Develop a CNN for the "tell-the-time" problem

Task 1: Learn basics of Keras and TensorFlow.

Start with reading the section "Implementing MLPs with Keras" from Chapter 10 of Geron's textbook (pages 295-327). Then install TensorFlow 2.0 (you will find some instructions on the Blackboard) and experiment with the code included in this section. Study the "semi-official" documentation of Keras (https://keras.io/) and get an idea of the numerous options offered by Keras (layers, loss functions, metrics, optimizers, activations, initializers, regularizers). Don't get overwhelmed with the number of options – you will frequently return to this side in the coming months.

There is a great website with many examples of Keras implementations of various sorts of networks: https://github.com/keras-team/keras/tree/master/examples . Visit this site (or just clone this directory to your computer) and try to get some of these examples running on your system. In particular, experiment with mnist_mlp.py and mnist_cnn.py programs. Note, that all these programs have been created "before" TensorFlow 2.x so they will not immediately work on your system. You will have to modify them slightly (as described in your textbook, Ch. 10) to get them running. The skill of adjusting Keras code to TF 2.x is very important: most of Keras code that is available on the internet has been created before the TF 2.x era! (Just to remind you: before TF 2.0 Keras was an API that was supported by several systems like Theano, TensorFlow 1.x, CNTK. TF 2.x uses Keras as its internal API.) There are at least two aspects that play a role when converting older code to TF 2.x:

(1) How Keras is imported to your program: this aspect is explained in your textbook, Ch. 10,
(2) How color images, batches of images, multiple feature maps are represented? E.g., a color image of size 10x10 can be represented by a 3x10x10 tensor ("channels_first"; default for Theano backend) or a 10x10x3 tensor ("channels last"; default for Tensorflow backend). This aspect is explained in Ch 11, pp. 451). You can get more information on this issue at https://machinelearningmastery.com/a-gentle-introduction-to-channels-first-and-channels-last-image-formats-for-deep-learning/

After getting TF 2.x working on your system you can start "real work". Take the two well-known datasets: MNIST (introduced in Ch. 3, p. 85) and Fashion MNIST (introduced in Ch 10, p. 297). Both

datasets contain the same number of 2D images of size 28x28, split into 10 categories; 60.000 images for training and 10.000 for testing. Apply two reference networks to both datasets: a MLP described in detail in Ch. 10, pp. 297-307 and a CNN described in Ch. 14, p. 461. Experiment with both networks, trying various options: initializations, activations, training algorithms (and their hyperparameters), regularizations (L1, L2, Dropout, no Dropout). You may also experiment with changing the architecture of both networks: adding/removing layers, number of convolutional filters, their sizes, etc.

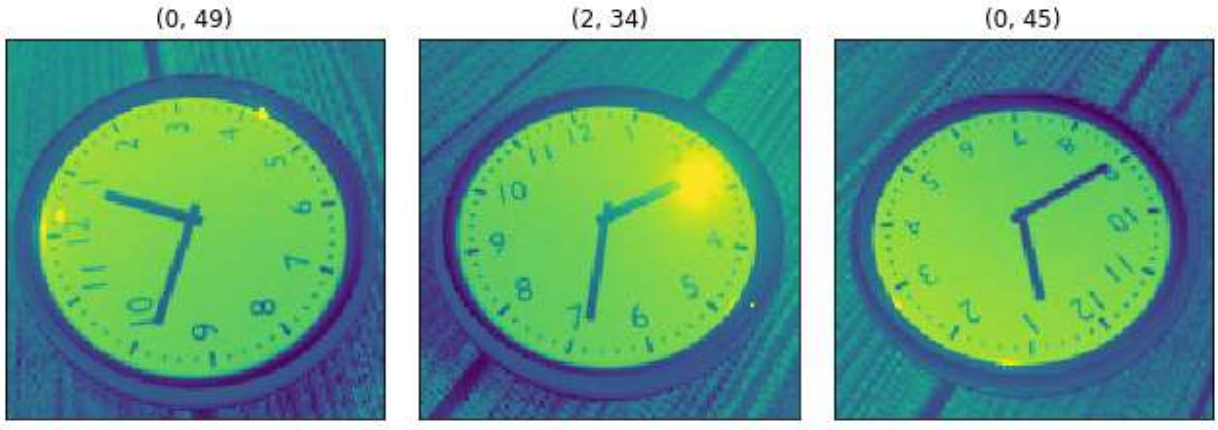
The purpose of this task is NOT to get the highest accuracy (which is usually the most important performance measure). Instead you are supposed to gain some practical experience with tuning networks, running multiple experiments (with help of some scripting), and, very important, documenting your findings. In your report you have to provide a concise description of your experiments, results, conclusions. The quality of your work (code and report) is more important than the quantity or the accuracy you've achieved.

Task 2: Testing the impact of obfuscating data by randomly permutating all pixels

After completing Task 1 select the most successful setups of your MLP and CNN networks. Next, modify both sets (MNIST and Fashion MNIST) by permuting all 28x28 pixels of all images with help of a single random permutation of all 784 pixels of both data sets. Retrain your both networks on permuted versions of the training sets and report accuracies on the test sets. In theory, the accuracies of MLP networks should be almost the same as those trained on the unpermuted data; convolutional networks should perform much worse. Is it really the case? Please, comment on your findings.

Task 3: Develop a "Tell-the-time" network.

We have produced a big collection of images of a traditional clock and numeric representation of the time shown on these images. Your task it to develop a DNN that, when applied to images from the test set would tell the time as correctly as possible. Here is a random sample of 3 images from our collection with the corresponding time (hour, minutes) printed on top of these images:



The dataset can be downloaded from https://surfdrive.surf.nl/files/index.php/s/B8emtQRGUeAaqmz (~300MB) and it consists of 18000 grayscale images (18000x150x150) contained in *'images.npy'*. The

labels for each sample are represented by two integers (18000x2, *'labels.npy' file*), that correspond to the hour and minute displayed by the clock.

The goal of this task is two-fold:

1. The problem of correctly telling the time can be formulated either as a multi-class classification problem (for example, with 12x60 classes) or a regression problem (for example, predicting the number of minutes after 12 o'clock). Therefore, try to come up with different representations for the labels of your data, adapting the loss function and output layer of your neural network and see how it impacts the training time and performance. No matter which architecture and loss function you will use, when reporting results also provide "common sense" accuracy: the absolute value of the time difference between the predicted and the actual time (e.g., the "common sense" difference between "predicted" 11:55 and the "target" 0:05 is just 10 minutes and not 11 hours and 50 minutes!). Minimizing this "common sense" error measure is the main objective of this assignment! Notice that it is a common situation in Machine Learning: we often train models using one error measure (e.g., cross-entropy) while the actual error measure that we are interested in is different, e.g., the accuracy (the percentage of correctly classified cases).

2. Use the knowledge gained by working with other datasets in the previous parts of this assignment to optimize your models and decrease the error of telling the time as much as possible (accuracy of below 10 minutes is achievable using relatively simple CNN architectures).

You should use a 80:20% ratio for the train and test sets. Document your experiments and findings in the report.